Exemple d'interface graphique pour les factorisations matricielles QR et LU avec PyQT6, Scipy et Numpy

Peter Philippe

13/06/2025

Introduction

La factorisation QR est un remarquable algorithme itératif et fondamental en algèbre linéaire. Il effectue la décomposition d'une matrice $\mathbf{A} \in \mathbb{R}^{m \times n}$ (avec $m \geqslant n$, mais on ne s'occupera que du cas où $n = m$), dont le rang n'est pas nécessairement plein, en un produit, **unique**, d'une matrice orthogonale $\mathbf{Q} \in \mathbb{R}^{m \times n}$ possédant la remarquable propriété suivante : $\mathbf{Q}\mathbf{Q}^{\mathrm{T}} = \mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{Q}\mathbf{Q}^{-1} = \mathbf{Q}^{-1}\mathbf{Q} = \mathbf{I}$, avec $|\mathbf{Q}| = \pm 1$ et dont le nombre de conditionnement vaut $\kappa(\mathbf{Q}) = 1$) et d'une matrice triangulaire supérieure régulière $\mathbf{R} \in \mathbb{R}^{n \times n}$ (tous les éléments de la diagonale, donc les valeurs propres, représentent le spectre de $\mathbf{R}$) telle que $\mathbf{A} = \mathbf{QR}$. Ceci permet notamment de résoudre un système linéaire de type $\mathbf{Ax} = \mathbf{b}$ tel que $\mathbf{QRx} = \mathbf{b}$ avec $\mathbf{y} = \mathbf{Q}^{\mathrm{T}}\mathbf{b} = \mathbf{Q}^{-1}\mathbf{b}$ puis $\mathbf{Rx} = \mathbf{y}$ (pour la méthode des moindres carrés par exemple), de calculer les valeurs propres $\mathbf{Ax} = \lambda\mathbf{x}$ (ici λ est une valeur propre associée à son vecteur propre $\mathbf{x}$).

Plusieurs manières existent pour calculer cette factorisation, notamment avec la transformation de Householder [1]. Cet algorithme est d'ailleurs proposé par Numpy [2] avec la fonction `np.linalg.qr()`, il s'agit en fait d'un appel à la fonction `dgeqrf()` en langage Fortran de la librairie LAPACK.

1. Alston Scott Householder (1904-1993) - Mathématicien américain
2. https ://numpy.org

Au cours des dernières décennies, différentes améliorations ont été apportées à cette méthode, en particulier avec la méthode décalée permettant d'améliorer significativement la précision au fil des itérations, ainsi que la méthode *double QR* adaptée aux matrices dont les valeurs propres sont toutes complexes.

En ce qui concerne la factorisation LU, elle est utilisée pour la résolution d'un système linéaire de type $\mathbf{Ax} = \mathbf{b}$ via la résolution de deux autres systèmes triangulaires $\mathbf{Ly} = \mathbf{b}$ et $\mathbf{Ux} = \mathbf{y}$, mais également en tant que préconditionneur ILU(0) pour des matrices non symétriques, creuses ou non, avec des solveurs basés sur le processus de Lanczos (méthode MINRES par exemple) ou celui de Arnoldi (méthode GMRES entre autres), le processus de triangularisation (ou de trigonalisation) reprend celui de la méthode du pivot de Gauss.

Ci-dessous, il y aura deux très petits exemples détaillant étape par étape ces deux factorisations pour une seule et même matrice carrée $\mathbf{A}$ d'ordre 3 :

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 2 & 5 & 4 \\ 1 & 7 & 3 \\ 1 & 4 & 6 \end{pmatrix}$$

Au passage, votre œil expert a immédiatement remarqué que :

$$a_{11} + a_{12} + a_{13} = a_{21} + a_{22} + a_{23} = a_{31} + a_{32} + a_{33} = 11$$

L'une des valeurs propres λ de $\mathbf{A}$ est donc égale à $\boxed{11}$, sa valeur propre correspondante sera toujours dans ce cas particulier $\mathbf{x} = (1{,}1{,}1)^{\mathrm{T}}$. Cette astuce de calcul pour la valeur propre est aussi valable pour les colonnes, en revanche, le vecteur propre sera à calculer.

1 Exemple de factorisation LU avec la version de Crout

Pour une matrice d'ordre 3, les opérations de cette factorisation sont les suivantes :

$$\mathbf{A} = \mathbf{LU}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & u_{3,3} \end{pmatrix}$$

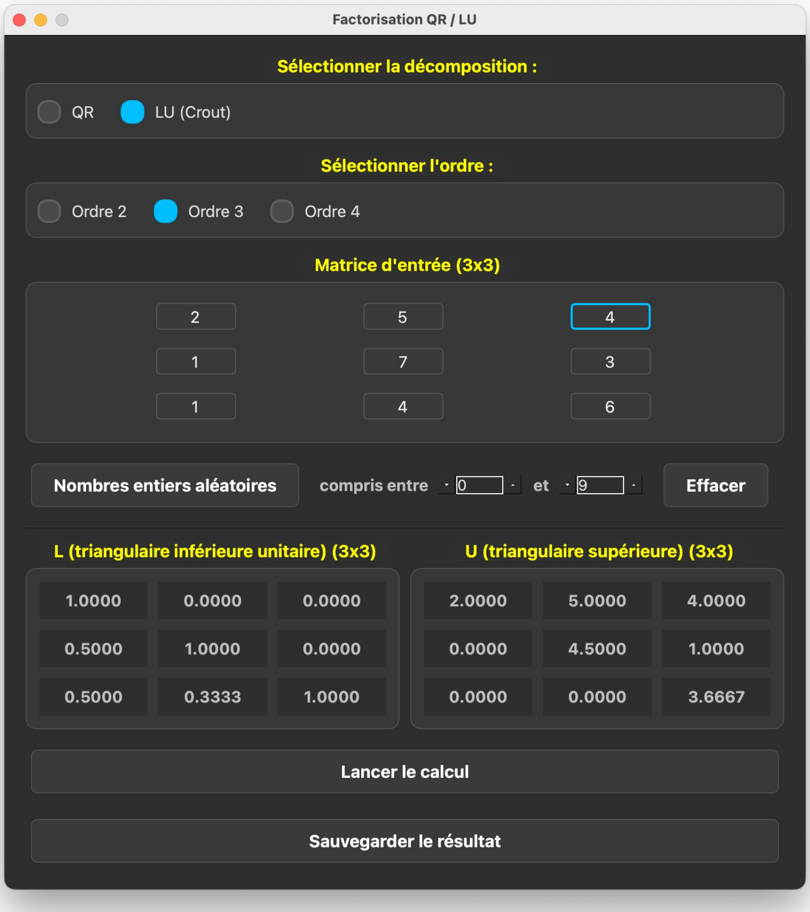
$$= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{2,1}}{u_{1,1}} & 1 & 0 \\ \frac{a_{3,1}}{u_{1,1}} & \frac{a_{3,2}-l_{3,1}u_{1,2}}{u_{2,2}} & 1 \end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,2} - l_{2,1}u_{1,2} & a_{2,3} - l_{2,1}u_{1,3} \\ 0 & 0 & a_{3,3} - l_{3,1}u_{1,3} - l_{3,2}u_{2,3} \end{pmatrix}$$

Soit :

$$\begin{pmatrix} 2 & 5 & 4 \\ 1 & 7 & 3 \\ 1 & 4 & 6 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & \frac{4-\frac{1}{2}\times 5}{7-\frac{1}{2}\times 5} = \frac{1{,}5}{4{,}5} & 1 \end{pmatrix} \begin{pmatrix} 2 & 5 & 4 \\ 0 & 7 - \frac{1}{2} \times 5 & 3 - \frac{1}{2} \times 4 \\ 0 & 0 & 6 - \frac{1}{2} \times 4 - \frac{1{,}5}{4{,}5} \end{pmatrix}$$

$$= \boxed{\begin{pmatrix} 1 & 0 & 0 \\ 0{,}5 & 1 & 0 \\ 0{,}5 & 0{,}333\ldots & 1 \end{pmatrix} \begin{pmatrix} 2 & 5 & 4 \\ 0 & 4{,}5 & 1 \\ 0 & 0 & 3{,}666\ldots \end{pmatrix}}$$

Ce qui est confirmé par notre petit outil :



2 Algorithme de la factorisation de Householder

On rappelle que la matrice de Householder est carrée, symétrique et de rang maximal :

$$\begin{aligned}\mathbf{H}(\mathbf{x}) &= \mathbf{I} - 2\frac{\mathbf{x}\,\mathbf{x}^T}{\mathbf{x}^T\mathbf{x}} \\ &= \begin{pmatrix} 1-2\mathbf{x}_1^2 & 2\mathbf{x}_1\mathbf{x}_2 & \dots & -2\mathbf{x}_1\mathbf{x}_n \\ -2\mathbf{x}_2\mathbf{x}_1 & 1-2\mathbf{x}_2^2 & \dots & -2\mathbf{x}_2\mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ -2\mathbf{x}_n\mathbf{x}_1 & -2\mathbf{x}_n\mathbf{x}_2 & \dots & 1-2\mathbf{x}_n^2 \end{pmatrix}\end{aligned}$$

Le vecteur $\mathbf{x}$ vaut $\mathbf{x} = \mathbf{a} \pm \|a\|_2 e_1$ où $\mathbf{a}$ représente la première colonne de $\mathbf{A}_n$, a sont les coefficients de cette colonne, $\|\cdot\|_2$ est la norme euclidienne c'est-à-dire $\sqrt{\sum_i a_i}$, enfin e_1 est un vecteur (colonne) dont l'ordre est celui de $\mathbf{H}_n$ et son premier coefficient vaut toujours 1, les autres sont nuls.

Comme l'ordre de $\mathbf{A}$ vaut $n = 3$, la formule pour obtenir $\mathbf{Q}$ est

$$\mathbf{Q} = \mathbf{H}_1\mathbf{H}_2\mathbf{H}_3$$

et celle pour $\mathbf{R}$ (attention à l'ordre des produits) :

$$\mathbf{R} = \mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{A}$$

On rappelle que $\mathbf{A}$ est :

$$\mathbf{A} = \begin{pmatrix} \mathbf{2} & 5 & 4 \\ \mathbf{1} & 7 & 3 \\ \mathbf{1} & 4 & 6 \end{pmatrix}$$

Posons $\mathbf{A} = \mathbf{A}_1$, puis débutons par le calcul de la matrice $\mathbf{Q}$. Tout d'abord, il nous faut déterminer $\mathbf{x}_1$ (qui requiert donc la première colonne de $\mathbf{A}_1$) :

$$\mathbf{x}_1 = \begin{pmatrix} \mathbf{2} \\ \mathbf{1} \\ \mathbf{1} \end{pmatrix} + \sqrt{\mathbf{2}^2 + \mathbf{1}^2 + \mathbf{1}^2} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2+\sqrt{6} \\ 1 \\ 1 \end{pmatrix}$$

Puis celui de $\mathbf{H}_1$ du même ordre que $\mathbf{A}_1$ soit 3 :

$$\mathbf{H}_1(\mathbf{x}_1) = \mathbf{I}_3 - 2\frac{\begin{pmatrix} 2+\sqrt{6} & 1 & 1 \end{pmatrix}\begin{pmatrix} 2+\sqrt{6} \\ 1 \\ 1 \end{pmatrix}}{\begin{pmatrix} 2+\sqrt{6} \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 2+\sqrt{6} & 1 & 1 \end{pmatrix}} = \boxed{\begin{pmatrix} -0.8164966 & -0.4082483 & -0.4082483 \\ -0.4082483 & 0.9082483 & -0.0917517 \\ -0.4082483 & -0.0917517 & 0.9082483 \end{pmatrix}}$$

Le produit $\mathbf{H}_1\mathbf{A}_1$ nous permet d'éliminer tous les coefficients sous a_{11} de la matrice $\mathbf{H}_1\mathbf{A}_1$ et obtenir ainsi une première sous-matrice $\mathbf{A}_2$ (en gras) d'ordre 2 :

$$\mathbf{H}_1\mathbf{A}_1 = \begin{pmatrix} -0.8164966 & -0.4082483 & -0.4082483 \\ -0.4082483 & 0.9082483 & -0.0917517 \\ -0.4082483 & -0.0917517 & 0.9082483 \end{pmatrix} \begin{pmatrix} 2 & 5 & 4 \\ 1 & 7 & 3 \\ 1 & 4 & 6 \end{pmatrix}$$

$$= \begin{pmatrix} -2.4494897 & -8.5732141 & -6.9402209 \\ 0 & \mathbf{3.9494897} & \mathbf{0.5412415} \\ 0 & \mathbf{0.9494897} & \mathbf{3.5412415} \end{pmatrix}$$

Notre matrice $\mathbf{A}_2$ d'ordre 2 est donc :

$$\mathbf{A}_2 = \begin{pmatrix} \mathbf{3.9494897} & 0.5412415 \\ \mathbf{0.9494897} & 3.5412415 \end{pmatrix}$$

Le calcul de $\mathbf{x}_2$ réclame la première colonne de la matrice $\mathbf{A}_2$:

$$\mathbf{x}_1 = \begin{pmatrix} \mathbf{3.9494897} \\ \mathbf{0.9494897} \end{pmatrix} + \sqrt{\mathbf{3.9494897}^2 + \mathbf{0.9494897}^2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 8.0115089 \\ 0.9494897 \end{pmatrix}$$

S'en suit le calcul de $\mathbf{H}_2$ d'ordre 2 :

$$\mathbf{H}_2(\mathbf{x}_2) = \mathbf{I}_2 - 2\frac{\begin{pmatrix} 8.0115089 & 0.9494897 \end{pmatrix} \begin{pmatrix} 8.0115089 \\ 0.9494897 \end{pmatrix}}{\begin{pmatrix} 8.0115089 \\ 0.9494897 \end{pmatrix} \begin{pmatrix} 8.0115089 & 0.9494897 \end{pmatrix}} = \boxed{\begin{pmatrix} -0.9722972 & -0.2337482 \\ -0.2337482 & 0.9722972 \end{pmatrix}}$$

Le produit $\mathbf{H}_2\mathbf{A}_1$ est :

$$\mathbf{H}_2\mathbf{A}_1 = \begin{pmatrix} -0.9722972 & -0.2337482 \\ -0.2337482 & 0.9722972 \end{pmatrix} \begin{pmatrix} 3.9494897 & 0.5412415 \\ 0.9494897 & 3.5412415 \end{pmatrix}$$

$$= \begin{pmatrix} -4.0620192 & -1.3540064 \\ 0 & \mathbf{3.3166248} \end{pmatrix}$$

d'où $\mathbf{A}_2 = \mathbf{3.3166248}$. Pour terminer, $\mathbf{x}_3$ est réduit à un nombre réel $\mathbf{x}_3 = \mathbf{A}_2 = 3.3166248$ puisque $\mathbf{x}_3 = 3.3166248 - \sqrt{3.3166248^2(1)} = 0$ ce qui est interdit. La matrice 1×1, ou plutôt le réel $\mathbf{H}_3$ est alors :

$$\mathbf{H}_3(\mathbf{x}_3) = 1 - 2\frac{3.3166248^2}{3.3166248^2}(1) = \boxed{-1}$$

Afin de pouvoir être multipliées, les matrices $\mathbf{H}_2$ et $\mathbf{H}_3$ doivent être « intégrées » (si on peut aller ça de la sorte) dans une matrice unitaire d'ordre 3 en respectant leurs indices, c'est-à-dire :

$$\mathbf{H}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \boxed{-0.9722972} & \boxed{-0.2337482} \\ 0 & \boxed{-0.2337482} & \boxed{0.9722972} \end{pmatrix} \text{ et } \mathbf{H}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \boxed{-1} \end{pmatrix}$$

La matrice orthogonale $\mathbf{Q}$ recherchée vaut

$$\mathbf{Q} = \begin{pmatrix} -0.8164966 & -0.4082483 & -0.4082483 \\ -0.4082483 & 0.9082483 & -0.0917517 \\ -0.4082483 & -0.0917517 & 0.9082483 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.9722972 & -0.2337482 \\ 0 & -0.2337482 & 0.9722972 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

$$= \boxed{\begin{pmatrix} -0.8164966 & 0.492366 & 0.3015114 \\ -0.4082483 & -0.8616405 & 0.3015113 \\ -0.4082483 & -0.1230915 & -0.9045341 \end{pmatrix}}$$

et la matrice triangulaire supérieure $\mathbf{R}$ est :

$$\mathbf{R} = \mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{A}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -0.9722972 & -0.2337482 \\ 0 & -0.2337482 & 0.9722972 \end{pmatrix} \begin{pmatrix} -0.8164966 & -0.4082483 & -0.4082483 \\ -0.4082483 & 0.9082483 & -0.0917517 \\ -0.4082483 & -0.0917517 & 0.9082483 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 5 & 4 \\ 1 & 7 & 3 \\ 1 & 4 & 6 \end{pmatrix}$$

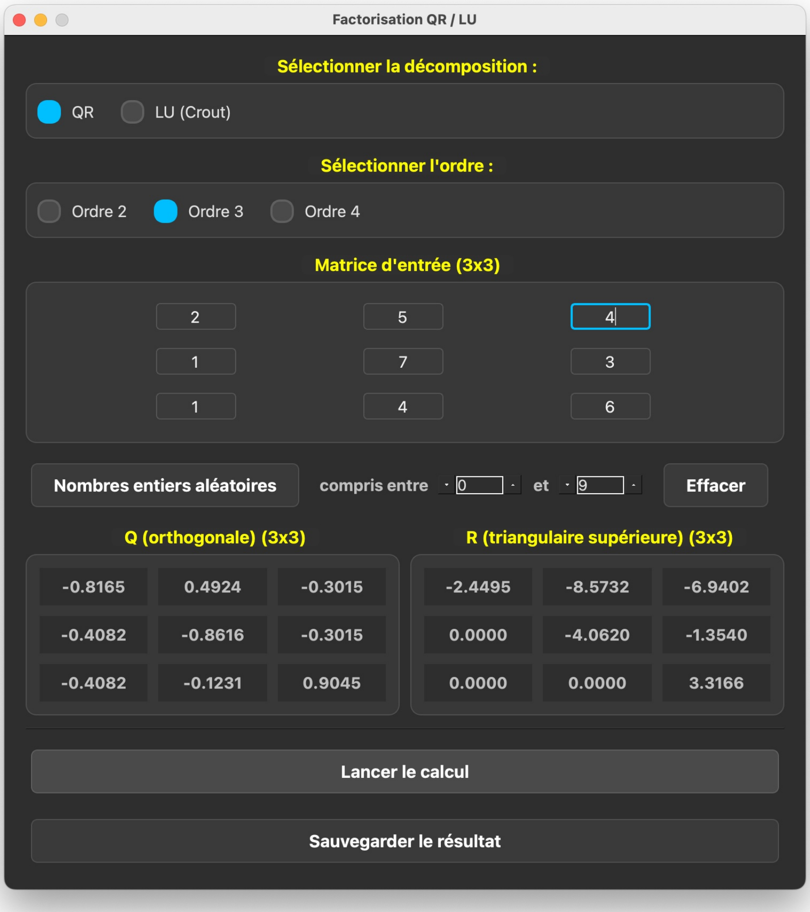
$$= \begin{pmatrix} -2.4494897 & -8.5732141 & -6.9402209 \\ 0 & -4.0620193 & -1.3540064 \\ 0 & -0.0000001 & -3.3166249 \end{pmatrix}$$

Au final, le produit $\mathbf{QR}$ permet de retrouver $\mathbf{A}$:

$$\mathbf{A} = \mathbf{QR}$$

$$\begin{pmatrix} 2 & 5 & 4 \\ 1 & 7 & 3 \\ 1 & 4 & 6 \end{pmatrix} = \boxed{\begin{pmatrix} -0{,}8164966 & 0{,}492366 & 0{,}3015144 \\ -0{,}4082483 & -0{,}8616405 & 0{,}3015144 \\ -0{,}4082483 & -0{,}1230915 & -0{,}9045341 \end{pmatrix} \begin{pmatrix} -2{,}4494897 & -8{,}5732141 & -6{,}9402209 \\ 0 & -4{,}0620193 & -1{,}3540064 \\ 0 & 0 & -3{,}3166249 \end{pmatrix}}$$

Là aussi, notre outil confirme ce résultat (où l'on remarque quelques différences de signes, mais le calcul final revient au même car on retrouve bien $\mathbf{A}$, ceci peut également se produire avec l'algorithme de Gram-Schmidt, qu'il soit modifié ou non) :



3 Implémentation en Python avec les librairies PyQT6, Scipy et Numpy

Le code source de ce petit programme, dont le code source est accessible à https://github.com/rayptor/linkedin/, il est suffisamment clair pour être compréhensible par toutes les personnes intéressées par PyQT :

```
# fichier factorisationQRLU.py

import sys
import numpy as np
import scipy.linalg as spla
from PyQt6.QtCore import Qt, QLocale, QCoreApplication, QRegularExpression
from PyQt6.QtGui import QFont, QRegularExpressionValidator
```

```python
from PyQt6.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QHBoxLayout, QSpinBox,
    QRadioButton, QLabel, QPushButton, QLineEdit, QGridLayout, QMessageBox,
    QGroupBox, QSizePolicy, QButtonGroup, QFrame, QFileDialog
)

class FactorisationQRLU(QWidget):
    def __init__(self) -> None:
        super().__init__()
        self.ordre: int = 3
        self.decomposition: str = "QR"
        self.entrees: list = []
        self.libelles_q: list = []
        self.libelles_r: list = []
        self.libelles_l: list = []
        self.libelles_u: list = []

        self.setWindowTitle("Factorisation QR / LU")
        self.setFixedSize(750, 800)
        self.setLayout(QVBoxLayout())
        self.setStyleSheet("""
           QWidget {
                background-color: #2E2E2E;
                color: #E0E0E0;
                font-size: 14px;
            }
            QPushButton {
                background-color: #3A3A3A;
                color: white;
                border: 1px solid #4F4F4F;
                border-radius: 6px;
                padding: 10px 20px;
                font-size: 16px;
                font-weight: bold;
                margin: 5px;
            }
            QPushButton:hover {
                background-color: #4A4A4A;
                border-color: #5F5F5F;
            }
            QPushButton:pressed {
                background-color: #2F2F2F;
                border-color: #3A3A3A;
            }
            QPushButton:focus {
                border: 2px solid #00BFFF;
            }
            QLabel {
                font-size: 15px;
                color: #C0C0C0;
                font-weight: bold;
```

```
    }
    QRadioButton {
        background-color: #383838;
        color: #E0E0E0;
        font-size: 15px;
        font-weight: normal;
        padding: 5px 0;
        margin-right: 15px;
    }
    QRadioButton::indicator {
        width: 18px;
        height: 18px;
        border-radius: 9px;
        border: 2px solid #606060;
        background-color: #4A4A4A;
        margin-right: 8px;
    }
    QRadioButton::indicator:hover {
        border: 2px solid #00BFFF;
    }
    QRadioButton::indicator:checked {
        background-color: #00BFFF;
        border: 2px solid #00BFFF;
    }
    QRadioButton::indicator:checked:hover {
        background-color: #00A3D9;
    }
    QLineEdit {
        background-color: #3A3A3A;
        color: #F0F0F0;
        font-size: 15px;
        border-radius: 4px;
        border: 1px solid #555555;
        padding: 2px;
        text-align: center;
    }
    QLineEdit:hover {
        border: 1px solid #777777;
    }
    QLineEdit:focus {
        border: 2px solid #00BFFF;
    }
    QSpinbox {
        font-size: 15px;
        color: #C0C0C0;
        font-weight: bold;
    }
    QSpinBox:hover
    {
        border: 1px solid #00BFFF;
    }
```

```
        QSpinBox::down-button {
            subcontrol-position: left;
            height: 16px;
            margin: 0 5px 1px 0;
            padding: 0 1px 1px 1px;
        }
        QSpinBox::up-button {
            subcontrol-position: right;
            height: 16px;
            margin: 0 5px 1px 0;
            padding: 0 1px 1px 1px;
        }
        QGroupBox {
            background-color: #383838;
            border: 1px solid #505050;
            border-radius: 10px;
            margin-top: 25px;
            font-size: 16px;
            font-weight: bold;
            color: yellow;
        }
        QGroupBox::title {
            subcontrol-origin: margin;
            subcontrol-position: top center;
            padding: 0 10px;
            background-color: #303030;
            border-radius: 5px;
        }
        .QRFactorizationApp QLabel {
            background: #333333;
            color: #FFFFFF;
            font-size: 13px;
            border: 1px solid #444444;
            border-radius: 4px;
            padding: 4px;
            qproperty-alignment: AlignCenter;
        }
    """)
    self.initialiser_matrice()

def initialiser_matrice(self) -> None:
    self.layout().addWidget(self._creer_boutons_radio_decomposition())
    self.layout().addWidget(self._creer_boutons_radio_ordre())

    self._matrice_layout = QGridLayout()
    self.matrice_groupbox = self._creer_groupbox("Matrice", self._matrice_layout)
    self.layout().addWidget(self.matrice_groupbox)

    hbox_layout = QHBoxLayout()
    hbox_layout.addWidget(self._creer_boutton("Nombres entiers aléatoires", \
                                              self.valeurs_aleatoires, 16))
```

```
label_compris_entre = QLabel("compris entre")
label_compris_entre.setAlignment(Qt.AlignmentFlag.AlignCenter)
hbox_layout.addWidget(label_compris_entre)

self.min_spinbox = QSpinBox()
self.min_spinbox.setRange(-99, 0)
self.min_spinbox.setValue(-9)
self.min_spinbox.setFixedWidth(80)
hbox_layout.addWidget(self.min_spinbox)

label_et = QLabel("et")
label_et.setAlignment(Qt.AlignmentFlag.AlignCenter)
hbox_layout.addWidget(label_et)

self.max_spinbox = QSpinBox()
self.max_spinbox.setRange(0, 99)
self.max_spinbox.setValue(9)
self.max_spinbox.setFixedWidth(80)
hbox_layout.addWidget(self.max_spinbox)

hbox_layout.addWidget(self._creer_boutton("Effacer", self.effacer_entrees, 16))
hbox_layout.addStretch()
self.layout().addLayout(hbox_layout)

self._q_layout = QGridLayout()
self.matrice_q_groupbox = self._creer_groupbox("Q", self._q_layout)
self._r_layout = QGridLayout()
self.r_groupbox = self._creer_groupbox("R", self._r_layout)

self._l_layout = QGridLayout()
self.l_groupbox = self._creer_groupbox("L", self._l_layout)
self._u_layout = QGridLayout()
self.u_groupbox = self._creer_groupbox("U", self._u_layout)

qr_matrices_layout = QHBoxLayout()
qr_matrices_layout.addWidget(self.matrice_q_groupbox)
qr_matrices_layout.addWidget(self.r_groupbox)
self.layout().addLayout(qr_matrices_layout)

ligne_separatrice = QFrame()
ligne_separatrice.setFrameShape(QFrame.Shape.HLine)
ligne_separatrice.setFrameShadow(QFrame.Shadow.Raised)
self.layout().addWidget(ligne_separatrice)

lu_matrices_layout = QHBoxLayout()
lu_matrices_layout.addWidget(self.l_groupbox)
lu_matrices_layout.addWidget(self.u_groupbox)
self.layout().addLayout(lu_matrices_layout)

self.matrice_q_groupbox.setVisible(self.decomposition == "QR")
```

```
        self.r_groupbox.setVisible(self.decomposition == "QR")
        self.l_groupbox.setVisible(self.decomposition == "LU")
        self.u_groupbox.setVisible(self.decomposition == "LU")

        self.layout().addWidget(self._creer_boutton("Lancer le calcul", \
                                            self.calculer_decomposition, 16, True))
        self.layout().addWidget(self._creer_boutton("Sauvegarder le résultat", \
                                            self.sauvegarder_resultats, 16, True))

        self.maj_entrees()
        self.maj_sorties()

    def _creer_boutons_radio_ordre(self) -> QGroupBox:
        box = QGroupBox("Sélectionner l'ordre :")
        layout = QHBoxLayout()
        self.groupe_boutons_ordre = QButtonGroup()

        for n in (2, 3, 4):
            bouton_radio = QRadioButton(f"Ordre {n}")
            bouton_radio.setProperty("ordre", n)
            bouton_radio.toggled.connect(self.changer_ordre)
            self.groupe_boutons_ordre.addButton(bouton_radio)
            layout.addWidget(bouton_radio)
            if n == self.ordre:
                bouton_radio.setChecked(True)
        layout.addStretch()
        box.setLayout(layout)
        return box

    def _creer_boutons_radio_decomposition(self) -> QGroupBox:
        box = QGroupBox("Sélectionner la décomposition :")
        layout = QHBoxLayout()
        self.groupe_boutons_decomposition = QButtonGroup()

        boutons_radio_qr = QRadioButton("QR")
        boutons_radio_qr.setProperty("type", "QR")
        boutons_radio_qr.toggled.connect(self.changer_decomposition)
        self.groupe_boutons_decomposition.addButton(boutons_radio_qr)
        layout.addWidget(boutons_radio_qr)
        if self.decomposition == "QR":
            boutons_radio_qr.setChecked(True)

        boutons_radio_lu = QRadioButton("LU (Crout)")
        boutons_radio_lu.setProperty("type", "LU")
        boutons_radio_lu.toggled.connect(self.changer_decomposition)
        self.groupe_boutons_decomposition.addButton(boutons_radio_lu)
        layout.addWidget(boutons_radio_lu)
        if self.decomposition == "LU":
            boutons_radio_lu.setChecked(True)

        layout.addStretch()
```

```python
        box.setLayout(layout)
        return box

    def _creer_groupbox(self, titre: str, layout: QGridLayout) -> QGroupBox:
        box = QGroupBox(titre)
        box.setLayout(layout)
        box.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Expanding)
        return box

    def _creer_boutton(
            self,
            libelle: str,
            callback,
            size: int = 10,
            bold: bool = False
            ) -> QPushButton:
        bouton_push = QPushButton(libelle)
        bouton_push.clicked.connect(callback)
        font = QFont()
        font.setPointSize(size)
        font.setBold(bold)
        bouton_push.setFont(font)
        return bouton_push

    def changer_ordre(self) -> None:
        bouton = self.groupe_boutons_ordre.checkedButton()
        if bouton and (ordre := bouton.property("ordre")) != self.ordre:
            self.ordre = ordre
            self.maj_entrees()
            self.maj_sorties()

    def changer_decomposition(self) -> None:
        bouton = self.groupe_boutons_decomposition.checkedButton()
        nouveau_type = bouton.property("type")
        if bouton and nouveau_type != self.decomposition:
            self.decomposition = nouveau_type
            self.matrice_q_groupbox.setVisible(self.decomposition == "QR")
            self.r_groupbox.setVisible(self.decomposition == "QR")
            self.l_groupbox.setVisible(self.decomposition == "LU")
            self.u_groupbox.setVisible(self.decomposition == "LU")
            self.maj_sorties()

    def maj_entrees(self) -> None:
        self._effacer_layout(self._matrice_layout)
        self.entrees.clear()
        validateur_regex = QRegularExpressionValidator(QRegularExpression(r"^[-]?\d*(\.\d*)?$"))
        for i in range(self.ordre):
            ligne = []
            for j in range(self.ordre):
                case_lineedit = QLineEdit("0")
                case_lineedit.setFixedSize(75, 25)
```

```python
            case_lineedit.setAlignment(Qt.AlignmentFlag.AlignCenter)
            case_lineedit.setValidator(validateur_regex)
            self._matrice_layout.addWidget(case_lineedit, i, j)
            ligne.append(case_lineedit)
        self.entrees.append(ligne)
    self.matrice_groupbox.setTitle(f"Matrice d'entrée ({self.ordre}x{self.ordre})")

def maj_sorties(self) -> None:
    output_data = (
        (self._q_layout, self.libelles_q, "Q (orthogonale)"),
        (self._r_layout, self.libelles_r, "R (triangulaire supérieure)"),
        (self._l_layout, self.libelles_l, "L (triangulaire inférieure unitaire)"),
        (self._u_layout, self.libelles_u, "U (triangulaire supérieure)")
    )

    for layout, libelles, title_suffix in output_data:
        self._effacer_layout(layout)
        libelles.clear()
        for i in range(self.ordre):
            ligne = []
            for j in range(self.ordre):
                libelle = QLabel("-")
                libelle.setAlignment(Qt.AlignmentFlag.AlignCenter)
                libelle.setMinimumSize(72, 28)
                libelle.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Expanding)
                layout.addWidget(libelle, i, j)
                ligne.append(libelle)
            libelles.append(ligne)
        if layout == self._q_layout:
            self.matrice_q_groupbox.setTitle(f"{title_suffix} ({self.ordre}x{self.ordre})")
        elif layout == self._r_layout:
            self.r_groupbox.setTitle(f"{title_suffix} ({self.ordre}x{self.ordre})")
        elif layout == self._l_layout:
            self.l_groupbox.setTitle(f"{title_suffix} ({self.ordre}x{self.ordre})")
        elif layout == self._u_layout:
            self.u_groupbox.setTitle(f"{title_suffix} ({self.ordre}x{self.ordre})")

def _effacer_layout(self, layout) -> None:
    while layout.count():
        enfant = layout.takeAt(0)
        if enfant.widget():
            enfant.widget().deleteLater()

def calculer_decomposition(self) -> None:
    try:
        matrice = np.array([[float(case.text()) for case in ligne] for ligne in self.entrees])
    except ValueError:
        QMessageBox.warning(self, "Erreur", "Entrée invalide dans la matrice.")
        return

    if self.decomposition == "QR" and np.isclose(np.linalg.det(matrice), 0):
```

```
            QMessageBox.warning(self, "Erreur", "La matrice est singulière pour la méthode QR.")
            return

        if self.decomposition == "LU" and np.isclose(np.linalg.det(matrice), 0):
            QMessageBox.warning(self, "Erreur", "La matrice est singulière pour la méthode LU.")
            return

        for i in range(self.ordre):
            for j in range(self.ordre):
                self.libelles_q[i][j].setText("-")
                self.libelles_r[i][j].setText("-")
                self.libelles_l[i][j].setText("-")
                self.libelles_u[i][j].setText("-")

        if self.decomposition == "QR":
            try:
                Q, R = spla.qr(matrice)
                for i in range(self.ordre):
                    for j in range(self.ordre):
                        self.libelles_q[i][j].setText(f"{Q[i, j]:.4f}")
                        self.libelles_r[i][j].setText(f"{R[i, j]:.4f}")
            except spla.LinAlgError as e:
                QMessageBox.warning(self, "Erreur QR", f"Décomposition QR impossible : {e}")
                return
        elif self.decomposition == "LU":
            try:
                _, L, U = spla.lu(matrice) # on ne s'occupe pas de la matrice de permutation ici
                for i in range(self.ordre):
                    for j in range(self.ordre):
                        self.libelles_l[i][j].setText(f"{L[i, j]:.4f}")
                        self.libelles_u[i][j].setText(f"{U[i, j]:.4f}")
            except spla.LinAlgError as e:
                QMessageBox.warning(self, "Erreur LU", f"Décomposition LU impossible : {e}")
                return

    def sauvegarder_resultats(self) -> None:
        matrices_a_sauvegarder = {}
        if self.decomposition == "QR":
            matrices_a_sauvegarder["Q"] = self._recuperer_matrice_depuis_libelles(self.libelles_q)
            matrices_a_sauvegarder["R"] = self._recuperer_matrice_depuis_libelles(self.libelles_r)
        elif self.decomposition == "LU":
            matrices_a_sauvegarder["L"] = self._recuperer_matrice_depuis_libelles(self.libelles_l)
            matrices_a_sauvegarder["U"] = self._recuperer_matrice_depuis_libelles(self.libelles_u)

        if all(matrix is None for matrix in matrices_a_sauvegarder.values()):
            QMessageBox.information(self, "Aucune donnée", "Veuillez d'abord lancer un calcul \
                                    avant de sauvegarder.")
            return

        nom_fichier, _ = QFileDialog.getSaveFileName(self, "Sauvegarder le résultat (.csv)", \
                                        "résultats.csv", "Fichiers CSV (*.csv);;Tous fichiers (*)")
```

```
        if nom_fichier:
            try:
                with open(nom_fichier, "w", newline="", encoding="utf-8") as fichier:
                    for nom, matrice in matrices_a_sauvegarder.items():
                        if matrice is not None:
                            fichier.write(f"--> Matrice {nom}\n")
                            np.savetxt(fichier, matrice, fmt="%.4f", delimiter=";", \
                                       newline="\n", comments="")
                            fichier.write("\n")
                QMessageBox.information(self, "Sauvegarde réussie", f"Les résultats ont été \
                                        sauvegardés dans :\n{nom_fichier}")
            except Exception as e:
                QMessageBox.critical(self, "Erreur de sauvegarde", f"Une erreur est survenue \
                                     lors de la sauvegarde du fichier :\n{e}")

    def _recuperer_matrice_depuis_libelles(self, liste_libelles: list) -> np.ndarray | None:
        if not liste_libelles or not liste_libelles[0]:
            return None

        if liste_libelles[0][0].text() == "-":
            return None

        matrix = np.zeros((self.ordre, self.ordre))
        try:
            for i in range(self.ordre):
                for j in range(self.ordre):
                    matrix[i, j] = float(liste_libelles[i][j].text())
            return matrix
        except ValueError:
            return None

    def valeurs_aleatoires(self) -> None:
        rng = np.random.default_rng()
        for ligne in self.entrees:
            for case in ligne:
                case.setText(str(rng.integers(self.min_spinbox.value(), self.max_spinbox.value())))

    def effacer_entrees(self) -> None:
        for ligne in self.entrees:
            for case in ligne:
                case.setText("0")
        for i in range(self.ordre):
            for j in range(self.ordre):
                self.libelles_q[i][j].setText("-")
                self.libelles_r[i][j].setText("-")
                self.libelles_l[i][j].setText("-")
                self.libelles_u[i][j].setText("-")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    main = FactorisationQRLU()
```

```
    main.show()
    sys.exit(app.exec())
```